Requirements Specification of an Automotive System with Hybrid Sequence Charts

Radu Grosu*, Ingolf Krüger and Thomas Stauner†

Institut für Informatik, Technische Universität München

D-80290 München, Germany

http://www4.in.tum.de/~{grosu,kruegeri,stauner}/

Abstract

In this position statement we outline the key ideas behind Hybrid Sequence Charts (HySCs) along an example system taken from the automotive industry. HySCs are a visual description technique for communication in hybrid systems inspired by the well known Message Sequence Charts syntax. However, they have a completely different semantic model that is well suited for the application domain of hybrid systems. As modeling example we consider a scenario taken from the specification of an electronic height control system, which is used to adjust the chassis level of a car. HySCs can be advantageously used in the early phases of the system development process. In particular, in the requirements capture phase they can help to improve the dialog between customers and application experts. HySCs complement existing formalisms like hybrid automata by focusing on the interaction between the system's components. A detailed presentation of HySCs together with their formal semantics is given in [6].

1 Introduction

Requirements engineering for embedded systems usually necessitates to take intrinsic properties of the system's environment into account. In many cases, e.g. in

*Currently on leave at School of Engineering and Applied Science, Department of Computer and Information Science, 200 South 33rd Street, Philadelphia, PA 19104-6389, USA. Email: grosu@saul.cis.upenn.edu.

†This work was supported with funds of the Deutsche Forschungsgemeinschaft under the Leibniz program within project SysLab, and under reference number Br 887/9 within the priority program *Design and design methodology of embedded systems.*

ABS systems of cars, the environment is characterized by discrete transitions between different modes of continuous behavior. From an abstract point of view even some parts of the system itself may exhibit a mixture of discrete *and* continuous behavior. As requirements engineering is typically performed on a high level of abstraction, requirements specification for such embedded systems necessitates *hybrid* description techniques, i.e. techniques which are able to specify both discrete and continuous dynamics. In further development steps the continuous properties of the system and its environment can then be transformed into timing requirements for the system. Note that hybrid systems generalize real time systems by considering further physical quantities apart from time.

While a considerable number of description techniques for the behavior of hybrid systems has been developed in recent years (see e.g. [5] as a starting point), little work has been done to directly visualize the interaction between the components of a hybrid system. Inspired by the beneficial role interaction based description techniques, like [7, 8, 3, 2], play in the requirements specification of telecommunication and, more generally, object-oriented systems, we developed hybrid sequence charts (HySCs) in the attempt to carry over these advantages to hybrid systems [6].

HySCs may be seen as relatives of trajectories and timing diagrams. Trajectories probably are the most basic approach to visualizing a system's evolution. They consist of drawing plots of the evolution of the system's variables over the time axis (Fig. 1, top left). A more abstract description is obtained if we partition the time axis for each variable into qualitatively equivalent intervals and give a predicate describing the evolution within the respective interval (Fig. 1, bottom left). The result resembles timing diagrams [1], which are widely used in hardware design, and constraint diagrams [4]. With HySCs we want to emphasize the sequence of qualitative states each component of a hybrid system traverses in a system run. We therefore make a further abstraction step and use only one partitioning of the time axis into qualitatively equivalent intervals for each component of a system. In each partition all the component's variables evolve as specified by some predicate.

As the syntax of HySCs we adapt a subset of MSC-96 [7]. Besides the intuitive notation, this has the further advantage that developers can use standard syntax-directed graphic editors for their specifications. The semantics of HySCs, however, significantly differs from the semantics proposed for MSCs. In the semantic model used for HySCs a hybrid system is regarded as consisting of a set of time-synchronously operating components, each encapsulating a private state and com-

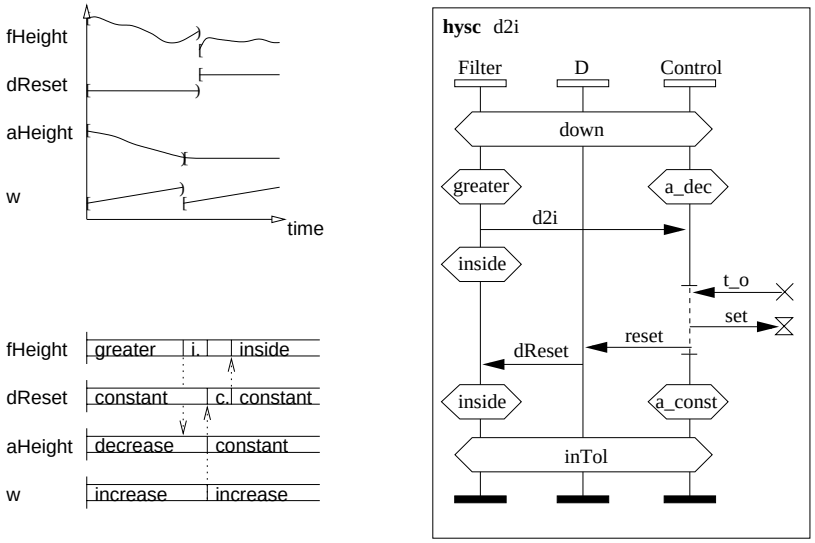

Figure 1: Description techniques for the interaction in hybrid systems.

municating with the other components over shared variables. The behavior of a component is characterized in this model by periods where the values of the variables change smoothly and by time instants at which there are discontinuities (see Fig. 1, top left). In our approach the discontinuities are caused by discrete actions. The smooth periods are caused by analog activities. The formal semantics of HySCs is given in [6].

In the following we sketch the main principles of HySCs along an example scenario of an *electronic height control system*. The purpose of this system is to keep the chassis level of a car within a prescribed tolerance interval by means of a pneumatic suspension system. The system is taken from a former case study together with BMW AG [9].

2 A Typical Scenario of the EHC

The scenario in Fig. 1, right, depicts the interaction in the EHC system when it changes from a state in which it had to decrease the chassis level to a state in which the chassis level is within the required bounds again.

The HySCs has one abstract time axis (the vertical lines in the figure) for each each of the considered components. Here, we consider a filter component `Filter`, a delay component `D`, and the controller component `Control`. The filter prepares the chassis level, as measured by sensors, for the controller. Depending on the filtered

chassis level the controller decides whether the chassis level has to be increased, decreased, or left constant during the next time interval between two expirations of its local timer. The delay component adds a certain delay to the signals the controller sends to the filter.

Before we explain the diagram in Fig. 1, right, we have to sketch some of the basic ideas behind HySC. In HySCs we use horizontal arrows to denote *events*. An event arrow is drawn whenever one of the variables of the component at the arrows origin reaches a value which may trigger a discrete action in the component at the arrows destination. As the execution of an action by a component may depend on the values of more than one of its input variables, a component usually receives a sequence of events, before it executes an action which possibly results in sending further events.

During the time intervals where no event occurs, all variables in the system evolve smoothly. Graphically we draw angular *condition boxes* which refer to predicates that describe the variables' evolution in the respective interval. Condition boxes may be local to one component or hierarchical, in which case they constrain the evolution of the variables of all those components whose instance axes they cover. A condition box remains valid up to the next condition box on the same level of hierarchy. Basically, condition boxes indicate the state of a component, or of a set of components, in the case of hierarchic conditions.

HySC `d2i` of Fig. 1, right, starts with global condition `down` which is refined into the local conditions `greater` and `a_dec` for the filter and the controller, respectively. Condition `down` characterizes the global system state in which the chassis level is decreased by the controller, `greater` indicates that the filtered chassis level is above the tolerance interval, and `a_dec` specifies that (and by what extent) the chassis level is being decreased. Furthermore, it states that the controller's timer has not yet expired.

With event `d2i` we denote that the filter value has just reached the tolerance interval. Condition `inside` determines that the value remains inside the interval. The next event arrow `t_o` refers to a predicate which denotes that the controller's timer has just expired. Together with the current value of the filtered chassis level being inside the tolerance interval this causes an action of the controller to be executed. The action results in the event `set` which resets the timer, and the event `reset` which signals the filter via the delay component that the filter's variables must be reset to certain predefined values.

After these events the filtered chassis level is still within the tolerance interval (condition `inside`), the controller is in a state where it does not modify the chassis level and waits for the next timer expiration (condition `a_const`). The global condition `inTol` refers to a predicate denoting that the controller leaves the chassis level unchanged.

We use coregions, visually depicted as dashed regions of an instance axis, to denote that the events within the region occur simultaneously. Thus, in our example the events `t_o`, `set` and `reset` occur at the same time. Furthermore, we use the timeout and set-timer symbols from MSC-96 in HySCs (see the arrows labeled `t_o` and `set`). We regard them as macros which are reduced to conditions and events over continuous variables, as explained in [6].

3 Conclusion and Outlook

In this position statement we have motivated interaction based description techniques for hybrid systems. Along an example scenario we informally introduced hybrid Sequence Charts (HySCs) as a concrete visual formalism for the communication in hybrid systems. HySCs syntax is inspired by the standardized syntax of MSC-96. Their semantics, however, is substantially different from standard MSCs.

A detailed description of HySCs is given in [6]. In order to specify the composition of HySCs, [6] additionally introduces *High-level HySCs* (HHSCs), whose syntax is also inspired in part from MSC-96. To make HHSCs applicable in the context of hybrid systems [6] furthermore provides notation for expressing preemption, which is an important concept for embedded systems.

References

[1] T. Amon, G. Borriello, T. Hu, and J. Liu. Symbolic timing verification of timing diagrams using presburger formulas. In *Proc. of the 34th Design Automation Conference*. ACM, 1997.

[2] M. Broy, C. Hofmann, I. Krüger, and M. Schmidt. A graphical description technique for communication in software architectures. Technical Report TUM-I9705, Technische Universität München, 1997.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A System of Patterns. Pattern-Oriented Software Architecture.* Wiley, 1996.

[4] C. Dietz. Graphical formalization of real-time requirements. In *Proc. FTRTFT'96*, LNCS 1135. Springer Verlag, 1996.

[5] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel. *Hybrid Systems.* LNCS 736. Springer-Verlag, 1993.

[6] R. Grosu, I. Krüger, and T. Stauner. Hybrid sequence charts. Technical Report TUM-I9914, Technische Universität München, 1999.

[7] ITU-TS. Recommendation Z.120 : Message Sequence Chart (MSC). Geneva, 1996.

[8] Unified modeling language, version 1.1. Rational Software Corporation, 1997.

[9] T. Stauner, O. Müller, and M. Fuchs. Using HyTech to verify an automotive control system. In *Proc. HART'97*, LNCS 1201. Springer-Verlag, 1997.